Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Analitzador de trames CAN FD basat en BitScope i Python

MEMÒRIA

Autor: Jaume Ribas Fernández
Director: Manuel Moreno Eguílaz

Convocatòria: Juny 2018

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

El CAN FD és un protocol de comunicació desenvolupat en els darrers anys i s'estima que serà el bus de comunicacions per excel·lència en la indústria de l'automoció durant els propers anys. Això és degut a les millores que incorpora respecte del seu predecessor, el bus CAN, les quals permeten transmetre més informació i de manera més ràpida. És per això que en aquest Treball de Fi de Grau s'ha desenvolupat un analitzador de trames CAN FD, fent servir el dispositiu BitScope Micro i el llenguatge de programació Python. Aquest dispositiu és d'un cost baix i mida reduïda, i suficientment eficaç per a l'obtenció de dades digitals. A més a més, permet d'una forma senzilla programar-lo en Python, i així s'ha implementat un algorisme per tractar les dades d'una trama de bus CAN FD i mostrar-les a través d'una interfície gràfica elaborada mitjançant el mòdul Tkinter de Python.

Sumari

1. Prefaci

1.1. Origen del projecte

Actualment, en el món de l'automoció es fa servir per a la transmissió de dades un protocol de comunicació anomenat CAN (*Controller Area Network*), que defineix l'estructura de la informació a transmetre, per tal de compartir dades entre les unitats electròniques de control (ECUs) que donen vida a un vehicle.

Aquest protocol s'ha millorat els darrers anys i, de fet, va ser a partir de l'any 2011 quan l'empresa Robert Bosch GmbH va començar a desenvolupar una millora d'aquest, anomenant-la CAN FD, que significa *CAN with Flexible Data-Rate*. Els aspectes més bàsics d'aquesta millora es defineixen per l'augment de la velocitat de transmissió de bits i l'augment de la quantitat d'informació possible a transmetre [1]. Com es pot intuir, aquestes característiques són de gran importància pel que interessa estudiar-ne el seu funcionament.

1.2. Motivació

A grans trets, es pot dir que aquest projecte neix de la inquietud de l'autor per l'electrònica en general i, més concretament, per aquells aspectes relacionats directament amb el *software* de dispositius electrònics, sense oblidar-ne també el seu *hardware*.

Per altra banda, existia també l'interès per part del tutor d'aquest treball de realitzar un projecte basat en l'esmentat ja protocol de comunicació CAN FD, per la seva probable aplicació en el món de l'automoció, del qual en forma part.

1.3. Requeriments previs

Per tal de poder realitzar aquest projecte, primer s'ha hagut d'estudiar el protocol de comunicació CAN, per després entendre el CAN FD i les seves millores, per tal de poder analitzar-lo i reproduir-lo gràficament.

Al mateix temps, s'ha estudiat el funcionament de l'analitzador de trames BitScope Micro proporcionat pel tutor, necessari per transformar la informació rebuda del CAN FD en valors

que es puguin emmagatzemar per representar posteriorment. D'aquesta forma, també s'ha estudiat com realitzar una interfície gràfica d'usuari per a fer aquesta representació.

2. Introducció

2.1. Objectius del projecte

El principal objectiu d'aquest projecte, tal i com diu el seu títol, és el d'aconseguir realitzar un analitzador de trames CAN FD, mitjançant l'ús d'un BitScope Micro i el llenguatge de programació Python. Al mateix temps, es pretén que es pugui implementar a una Raspberry Pi per tal de que esdevingui portàtil. La forma de l'ona es presentarà en una pantalla o projector, i per fer-ho s'usarà una interfície gràfica d'usuari, altrament coneguda com GUI (en anglès, *graphical user interface*).

D'aquest objectiu, se'n deriven d'altres, no per això menys importants, com conèixer el protocol CAN FD i aprendre a configurar una interfície gràfica d'usuari. A la vegada, tot i que no sigui un objectiu directe del treball però sí necessari, caldrà entendre el funcionament del BitScope Micro, per aprofitar-ne les seves utilitats.

Les possibles utilitats del projecte són molt diverses. Entre d'altres, les més destaques són:

- Eina didàctica per a estudiants d'automoció. En efecte, combinat amb un demostrador d'ECUs amb CAN FD existent en el Departament d'Electrònica, l'analitzador de trames desenvolupat en aquest projecte és l'eina perfecta per a mostrar als estudiants de forma interactiva i en temps real el funcionament del bus CAN FD.
- Eina de comprovació del funcionament d'un bus CAN FD. En efecte, l'analitzador de trames desenvolupat en aquest projecte és una eina molt útil per a comprovar el correcte funcionament dels nodes d'una xarxa de CAN FD.

2.2. Abast del projecte

Aquest projecte se centra en l'obtenció dels valors de trames CAN FD a diferents velocitats, el seu anàlisis i la seva representació gràfica, coses que es duran a terme a través de la programació en Python. Això és, aquí només es tractarà la part del *software*, i la del *hardware* ja vindrà donada pel tutor. És a dir, totes aquelles connexions amb cables necessàries pel projecte no es modificaran.

Per tal d'obtenir els valors de la trama caldrà configurar el BitScope Micro adientment, i així adquirir les mostres i identificar-les per mitjà d'un algorisme. Paral·lelament, s'haurà d'elaborar la interfície gràfica per tal que l'usuari hi pugui interactuar mínimament i se li mostri la trama

dibuixada i la seva informació.

2.3. Antecedents

Abans de posar-se a realitzar aquest projecte, s'ha realitzat una recerca d'altres projectes que poguessin resultar d'utilitat i d'aprofitament per aquest. Així doncs, cal dir que s'han trobat alguns en què s'utilitzava el mateix analitzador de trames però per un objectiu diferent. Al mateix temps, el propi fabricant del dispositiu subministra un programa base, tant en llenguatge Python com en C, amb l'estructura definida i amb una explicació dels passos a seguir, de forma que només calgui configurar-lo pel cas requerit i s'obtenen així les dades.

En el web del fabricant [2] també es disposa d'analitzadors de trames tant analògiques com digitals i oscil·loscopis, que, d'una forma senzilla, permeten a l'usuari visualitzar la forma d'aquestes en els diferents canals possibles del dispositiu. Un d'ells, el més proper al present projecte, és el BitScope Logic que es pot veure en la Figura 2.1, que analitza trames lògiques i té la capacitat d'identificar alguns protocols, entre ells el CAN, però no el CAN FD. Aquest *software* permet observar el comportament de diferents canals lògics a la vegada, en concret vuit, i també fins a quatre analògics. Entre les seves funcionalitats destaquen la possibilitat d'establir una condició per a l'inici de captació de dades, anomenada *trigger*, i la possibilitat d'obtenir valors per mitjà dels cursors, a part de la ja esmentada descodificació de protocols com, per exemple, I2C, SPI i CAN. Com ja s'explicarà més endavant, en determinats moments del projecte s'ha fet ús d'aquest analitzador lògic.

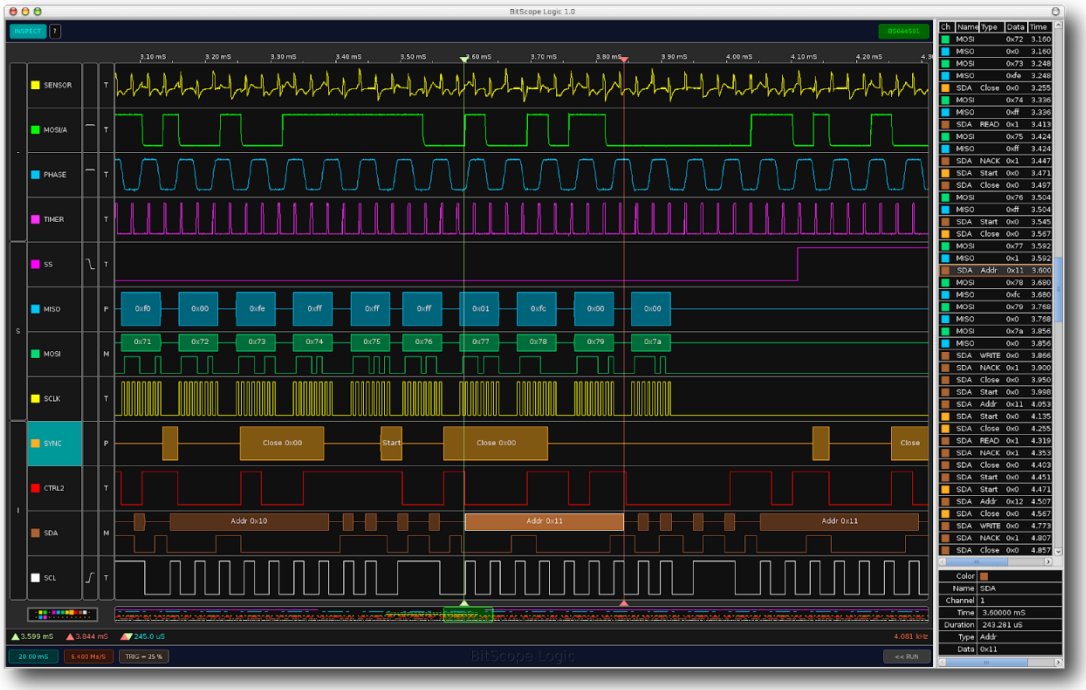

Figura 2.1 Interfície del software BitScope Logic. Font: [2].

Per altra banda, pel que fa a la interfície gràfica, també s'ha fet una recerca intensiva per tal d'intentar aprofitar-ne alguna ja realitzada. No s'ha trobat cap que fes exactament el que es volia, però s'han aprofitat parts d'algunes, com per exemple, per realitzar el gràfic o els botons de la interfície gràfica.

3. CAN

3.1. Què és CAN?

Tal i com ja s'ha definit abans, es tracta d'un protocol de comunicacions estàndard basat en un bus sèrie asíncron i amb l'objectiu d'interconnectar diferents nodes intel·ligents del sistema. Això vol dir que es realitza una transmissió de bits uns darrere dels altres, en un format fix, amb diferents velocitats possibles, i no existeix un senyal de rellotge únic, sinó que cada node té el seu però estan configurats de manera que s'assoleixi la màxima sincronització possible.

El senyal que es transmet és únic, i consisteix en el diferencial dels dos cables del bus, CAN H (CAN HIGH) i CAN L (CAN LOW). Un node pot enviar en qualsevol moment un missatge a través del bus, si no és que aquest ja està ocupat en aquell moment. Això comporta conflictes, de manera que per tal d'evitar problemes elèctrics, s'estableix el valor 0 com a dominant i l'1 com a recessiu, de forma que el missatge que tingui més zeros en la part inicial, que s'anomena identificador, tindrà més prioritat. Aquests nodes, al mateix moment que transmeten estan rebent, tal que si un node observa que el missatge que li arriba és diferent del que està enviant avorta la transmissió i posa el valor recessiu. Així, quan el bus estigui lliure, tornarà a enviar el missatge.

3.2. CAN FD

Aquesta millora del CAN ve donada principalment per l'increment de la velocitat de transmissió i de la quantitat de dades a transmetre. Està dissenyada de tal forma que en un mateix sistema puguin coexistir tant el CAN com el CAN FD, això és, no cal fer cap modificació del *software* ni del *hardware* dels nodes. Cal indicar que existeixen diferents tipus de trames dins d'aquest protocol, però en el present projecte tan sols es tindran en compte les trames de dades.

Dins de les modificacions establertes, hi ha la introducció de dos bits de control de més, així com els polinomis CRC, per tal d'assegurar la correcta transmissió de la informació. A més a més, a diferència del CAN, poden haver-hi dues velocitats de transmissió diferents en el mateix missatge, una per la "Arbitration Phase" i l'altra per la "Data Phase", tot i que aquestes velocitats són fixes per un mateix sistema.

En aquest protocol hi ha diferents mesures de detecció i comprovació de la transmissió de dades, per tal d'aconseguir la màxima seguretat. Entre aquestes mesures, hi ha els ja

esmentats polinomis CRC, els “Stuff Bits” o la monitorització dels nodes, consistent en la comparació dels bits que un node transmet i els que rep del bus, ja present en el CAN. Les dues primeres es discerniran més endavant.

Dins del CAN FD, igual que succeeix amb el CAN, existeixen dos formats (base/estàndard i estès), els quals difereixen en longitud en els camps de “Arbitration Field” i “Control Field”. En el format base, l’identificador té una llargada d’onze bits, mentre que en l’estès és de vint-i-nou.

3.2.1. Trama de dades

La trama de dades conté set camps de bits diferents que a continuació es descriuen.

- **Start of Frame (SOF):** indica l’inici de la trama i consisteix en un únic bit dominant.
- **Arbitration Field:** tal i com s’ha dit anteriorment, aquest camp canvia pel format base o l’estès. En el base, consisteix en el Base Identifier i el bit r1 com es veu en la Figura 3.1. En l’estès, consisteix en l’identificador estès, el bit SRR, el bit IDE i el bit r1, visible en la Figura 3.2. L’identificador estès està format per dues parts, la primera és el Base Identifier i la segona el Identifier Extension.
    - **Base Identifier:** són onze bits que indiquen el tipus d’informació de la trama i la seva prioritat.
    - **Identifier Extension Flag (IDE):** serveix per indicar en quin format està la trama. En el format base, pren el valor dominant i pertany al Control Field. Pel format estès, pren el valor recessiu i pertany al Arbitration Field.
    - **Substitute Remote Request (SRR):** es transmet només en el format estès i pren el valor recessiu. Serveix per, en cas de col·lisió entre dues trames amb el mateix Base Identifier, té prioritat el format estès.
    - **Identifier Extension:** són divuit bits que només es transmeten en el format estès i indiquen el tipus d’informació de la trama.
    - **r1:** bit dominant reservat per futures ampliacions del protocol.

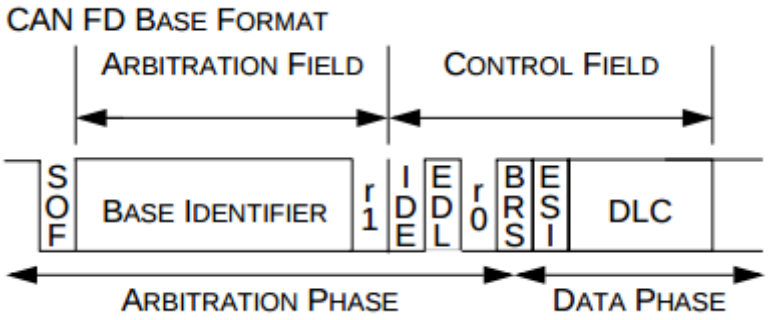

Figura 3.1 Identificació dels dos primers camps i el canvi de fase pel format base. Font: [1].

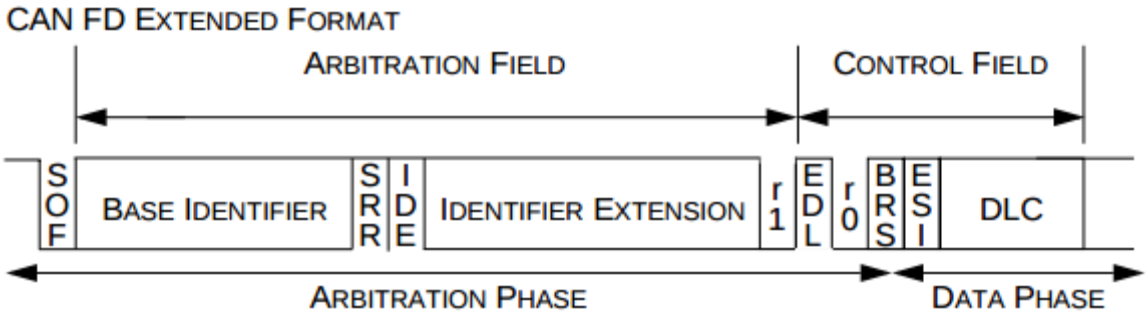

Figura 3.2 Identificació dels dos primers camps i el canvi de fase pel format estès. Font: [1].

- **Control Field:** la seva estructura també depèn del format. En el base, consisteix en els bits IDE, EDL, r0, BRS, ESI i els 4 del DLC, tal i com s'observa també en la Figura 3.1. En l'estès, consisteix en els EDL, r0, BRS, ESI, i els 4 del DLC, visible també en la Figura 3.2.
    - **Extended Data Length (EDL):** pren el valor recessiu i serveix per distingir entre una trama CAN FD i una CAN, la qual no té aquest bit. En el format base, precedeix el bit IDE, mentre que en l'estès es troba darrere del r1.
    - **r0:** bit dominant que sempre segueix el bit EDL i es reserva també per futures ampliacions del protocol.
    - **Bit Rate Switch (BRS):** serveix per indicar si hi ha dues velocitats de transmissió en la trama o només una. Si pren el valor recessiu, indica que la velocitat estàndard de l'Arbitration Phase canvia, a una altra sempre major, en la Data Phase, i si és dominant no es modifica. Si hi ha el canvi, es produeix a partir d'aquest mateix bit. La introducció d'aquest bit denota una millora rellevant respecte del CAN, el qual només permet una velocitat de fins a un 1 Mbit/s, i amb la millora, en la Data Phase es pot arribar fins a 8 Mbit/s. Per l'Arbitration Phase no és així, que té el mateix límit que el CAN.
    - **Error State Indicator (ESI):** és transmès dominant pels nodes *error active* i recessiu pels *error passive*.

- **Data Length Code (DLC):** el nombre de bytes en el camp Data Field ve donat pels valors d'aquests quatre bits. En el CAN ja existeixen, però amb la millora del CAN FD es permet major longitud del Data Field. La correlació ve indicada en la Taula 3.1.

	Number of Data Bytes	Data Length Code			
		DLC3	DLC2	DLC1	DLC0
Codes in CAN and CAN FD Format	0	0	0	0	0
	1	0	0	0	1
	2	0	0	1	0
	3	0	0	1	1
	4	0	1	0	0
	5	0	1	0	1
	6	0	1	1	0
	7	0	1	1	1
CAN Format	8	1	0/1	0/1	0/1
Codes in CAN FD Format	8	1	0	0	0
	12	1	0	0	1
	16	1	0	1	0
	20	1	0	1	1
	24	1	1	0	0
	32	1	1	0	1
	48	1	1	1	0
	64	1	1	1	1

Taula 3.1 Correlació dels bits del DLC amb la longitud del Data Field, tant pel format CAN com el CAN FD. Font: [1].

- **Data Field:** consisteix en les dades que es volen transmetre amb la trama. Tal i com es veu a la Taula 3.1, en el CAN FD es pot tenir fins a seixanta-quatre bytes de dades, una gran millora en comparació als vuit permesos en el CAN. Cada byte consisteix en vuit bits, els quals són transmesos per ordre de bit més significatiu (en anglès, *Most Significant Bit*). Si el valor de DLC és zero, aquest camp no existeix.
- **CRC Field:** aquest camp conté l'Stuff Count [3] i després la CRC Sequence. En aquest camp també es segueix l'ordre del bit més significatiu.
  - **Stuff Count:** conté un bit FSB, tres del Stuff Bit Counter i un de paritat.
    - **Fixed Stuff Bit (FSB):** pren el valor oposat de l'anterior bit. Pel CAN FD, la posició d'aquests bits en el CRC Field canvia respecte el CAN,

ja que, com ve diu el seu nom, estan en una posició fixa. A partir d'aquest primer, se situen després de cada quatre bits fins el CRC Sequence inclòs.

- **Stuff Bit Counter modulo 8:** abans del CRC Field, també hi ha Stuff Bits però en aquest cas dinàmics com el CAN. Això és, el node transmissor, quan detecta que envia cinc bits del mateix valor, hi afegeix a continuació un d'oposat, el que s'anomena *bit-stuffing*. Així doncs, perquè el node receptor pugui saber la longitud exacte de la trama cal que se li indiqui la quantitat de Stuff Bits dinàmics, i això es fa amb els tres bits del Stuff Bit Counter de mòdul 8 i codificat en codi Gray tal i com es pot veure en la Taula 3.2. Per assegurar la correcta transmissió d'aquesta rellevant informació pel protocol s'usen dues mesures de seguretat: el codi Gray ja esmentat, en el qual per indicar un Stuff Bit més només canvia el valor d'un bit del Stuff Bit Counter, i l'ús d'un bit de paritat.
- **Bit de Paritat:** bit que pren el valor indicat en la Taula 3.2 per cada Stuff Bit Counter associat.

Stuff bit count modulo 8	Bits added to FD frame format		
	Gray-coded value	Parity bit	Fixed stuff bit
0	000	0	1
1	001	1	0
2	011	0	1
3	010	1	0
4	110	0	1
5	111	1	0
6	101	0	1
7	100	1	0

Taula 3.2 Equivalència del nombre de Stuff Bits mòdul 8 amb els bits del Stuff Bit Counter en codi Gray, el bit de Paritat i el FSB següent. Font: [4].

- **CRC Sequence:** per comprovar que la trama és correcte s'utilitza el Cyclic Redundary Check (CRC), mètode consistent en la divisió de polinomis. Els

polinomis generadors de CRC són diferents pel CAN i pel CAN FD. A més a més, pel CAN FD depèn del valor del DLC: si és més petit o igual que 10 (16 bytes), la longitud del CRC és 17 sense comptar els FSB i el polinomi és el 0x3685B, i si és major que 10, la longitud és de 21 i el polinomi és el 0x302899. Al mateix temps, per realitzar el càlcul, el valor inicial també difereix, tal i com es pot observar en la Taula 3.3. Així doncs, el càlcul de comprovació que es realitza és la divisió de la trama obtinguda fins el Stuff Count inclòs pel polinomi generador, i el resultat ha de donar el mateix que els bits que s'obtenen posteriorment i que es poden veure en la Figura 3.3, que conformen el CRC Sequence. Tot això sense tenir en compte els FSB però sí els Stuff Bits dinàmics.

Valor del DLC	Longitud del Data Field	Longitud del CRC	Polinomi generador	Valor inicial
≤ 10	≤ 16 bytes	17 bits	0x3685B	0x1000
> 10	> 16 bytes	21 bits	0x302899	0x10000

Taula 3.3 Relació de valors per a la comprovació del CRC. Font: pròpia.

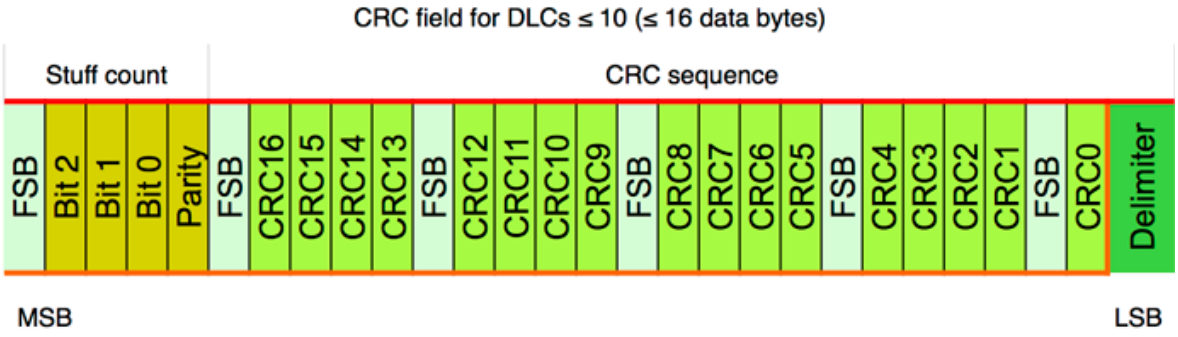

Figura 3.3 Situació dels bits en el Stuff Count i el CRC Sequence per un valor de DLC més petit o igual que 10. Font: [3].

- **CRC Delimiter:** consisteix en un o dos bits recessius. A partir d'aquest punt, es torna a tenir una Arbitration Phase, tal i com s'observa en la Figura 3.4, i la transmissió es fa novament a la velocitat inicial.

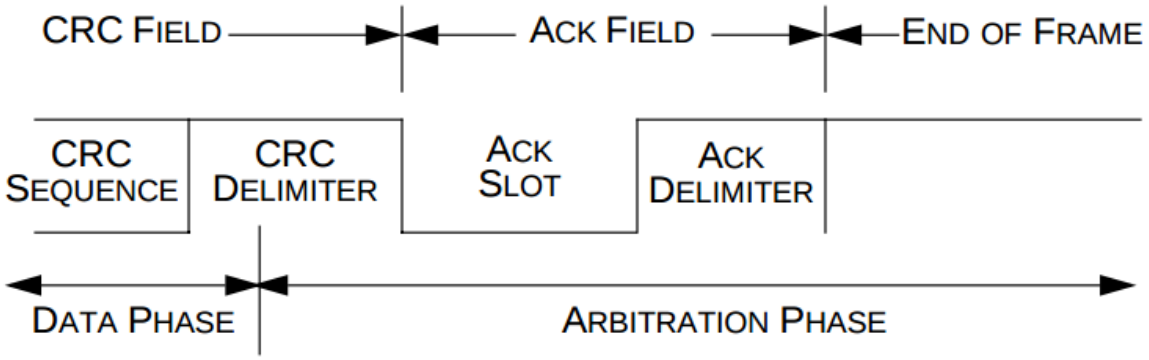

Figura 3.4 Identificació dels dos últims camps i el canvi de fase. Font: [1].

- **Acknowledge (ACK) Field:** conté el ACK Slot i el ACK Delimiter.
  - **ACK Slot:** si un node receptor rep un missatge correctament ho transmet al node emissor enviant-li un o dos bits dominants.
  - **ACK Delimite**r: consisteix en un bit recessiu.
- **End of Frame (EOF):** consisteix en set bits recessius que indiquen el final de la trama.

En les Figures 3.5, 3.6, 3.7 i 3.8 es poden observar les estructures gairebé senceres de les quatre trames possibles segons el format i el valor del DLC. Aquestes figures no inclouen els dos darrers camps ni tampoc els Stuff Bits, tant els dinàmics com els FSB.

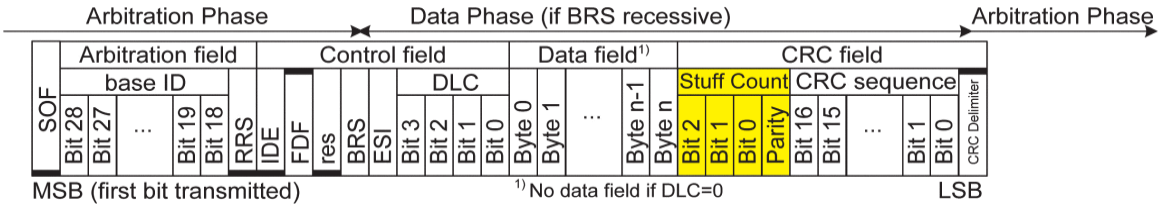

Figura 3.5 Posició dels bits pel format base i valor del DLC fins a 10. Font: [4].

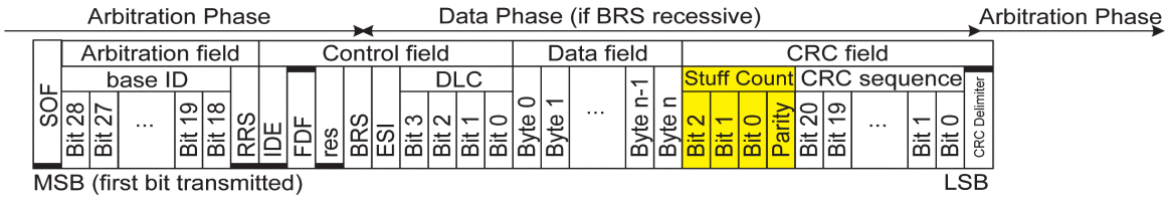

Figura 3.6 Posició dels bits pel format base i valor del DLC superior a 10. Font: [4].

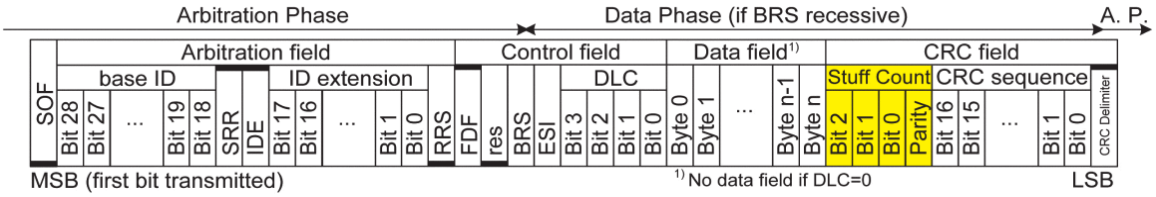

Figura 3.7 Posició dels bits pel format estès i valor del DLC fins a 10. Font: [4].

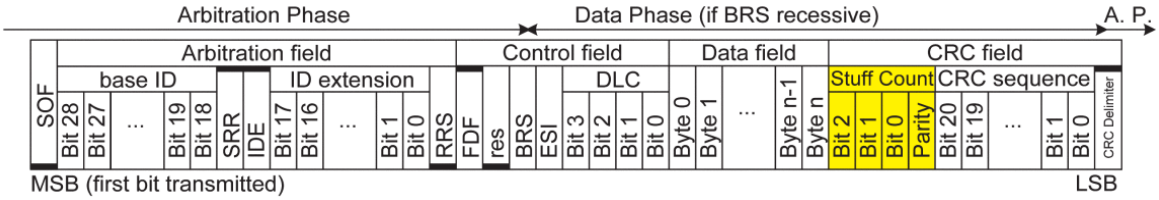

Figura 3.8 Posició dels bits pel format estès i valor del DLC superior a 10. Font: [4].

Cal indicar que, si de totes les comprovacions, algun bit no quadra, es dóna la trama per errònia. A la vegada, cal dir que en aquest projecte s'ha seguit en concret la darrera normativa ISO 11898-1 (CAN FD ISO), que resol uns problemes amb el CRC que es van detectar en l'inici de l'ús i estandardització del CAN FD [4].

A més a més, cal afegir que per tal de poder experimentar amb senyals en CAN FD, durant el projecte, el tutor ha proporcionat dues plaques que els reprodueixen. La primera, que es veu en la Figura 3.9, envia sempre el mateix senyal i de forma periòdica, i ha servit per la realització de la estructura base del codi final. La segona, que es correspon a la Figura 3.10 i és el resultat d'un altre projecte, permet transmetre diferents trames i a diferents velocitats, i ha servit per realitzar múltiples proves i acabar de configurar l'algoritme. Els senyals digitals que en surten de totes les plaques prenen els valors de 0 i 3,3 V.

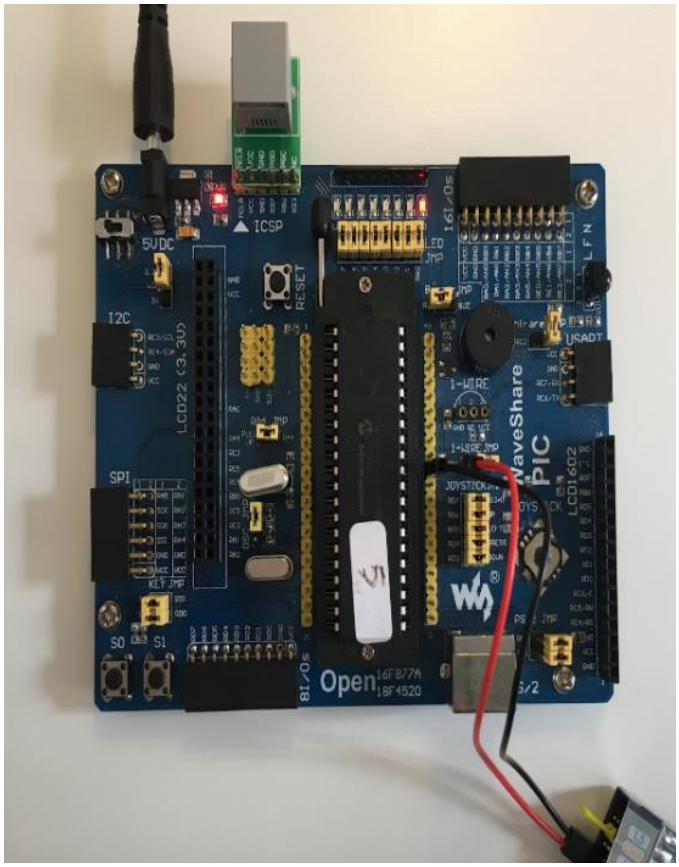

Figura 3.9 Placa que reprodueix una trama CAN FD periòdicament. Font: pròpia.

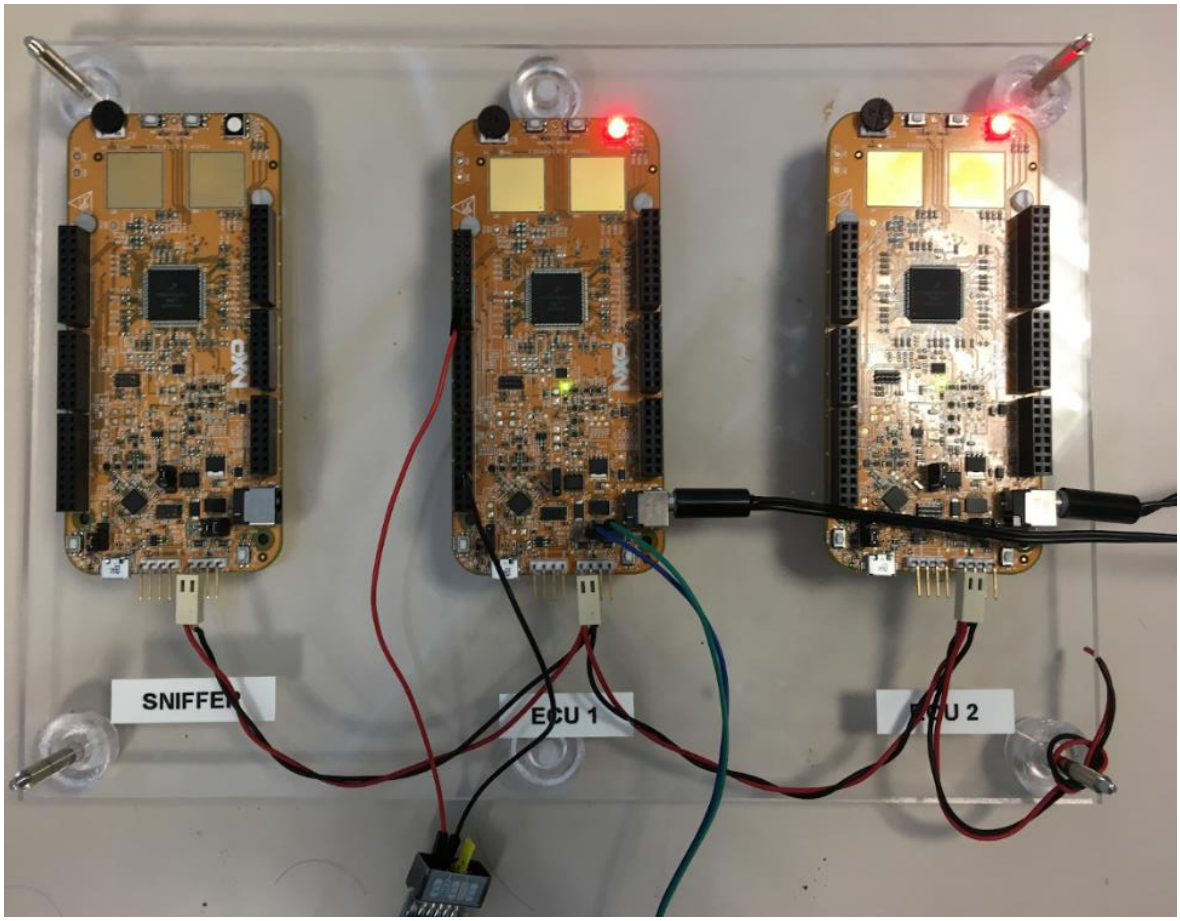

Figura 3.10 Placa que reprodueix múltiples trames CAN FD. Font: pròpia.

4. BitScope Micro (BS05U)

4.1. Introducció

El BitScope Micro (BS05U) és un dispositiu electrònic d'una mida reduïda que és capaç d'adquirir dades tant de senyals analògics, digitals o mixtos, amb les entrades adequades. S'alimenta connectant-lo a un port USB i es pot usar amb diferents sistemes operatius (Windows, Linux, Mac OS X) i, fins i tot, per Raspberry Pi [2], que consisteix en un petit ordinador integrat en una sola placa. Aquestes característiques, juntament amb la possibilitat de programar-lo en diferents llenguatges, fan que sigui de gran flexibilitat i, per tant, resulti molt útil. A més a més, en un principi va semblar que disposava d'unes especificacions que permetrien elaborar sense inconvenients el projecte desitjat, tot i que com es veurà més endavant, han sorgit una sèrie de problemes inesperats.

Aquest dispositiu també pot funcionar com a generador d'ones, rellotge o oscil·loscopi, ja que hi ha integrats els diferents components per fer-ho tal i com es pot veure en la Figura 4.1. Hi té inclosos també LEDs, i alguns d'ells són els que indiquen que està connectat (*Power LED*), que està preparat (*Data LED*) i que està mostrejant (*Sampling LED*).

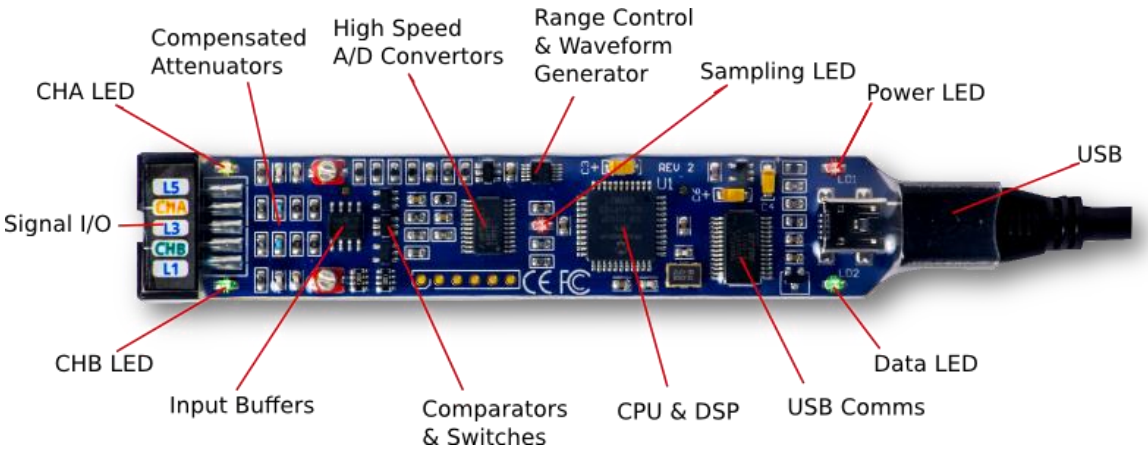

Figura 4.1 BitScope Micro (BS05U) amb indicacions dels seus components. Font: [2].

En termes més generals, BitScope és una empresa especialitzada en aquest tipus de dispositius electrònics per a adquisició de dades, i disposa de diferents *softwares* adaptats a aquests dispositius perquè l'usuari els pugui usar, o sinó, programar-los al seu gust. Per aquest motiu es proporciona a l'usuari també la biblioteca informàtica necessària per tal de dur-ho a terme. Aquest és el cas d'aquest projecte.

4.2. Especificacions tècniques

Les especificacions més rellevants d'aquest dispositiu són les que es descriuen a continuació en la Taula 4.1.

Característica	Especificacions
Màxim Ample de Banda analògic	20 MHz
Canals de captura	2 analògics i 6 lògics o 8 lògics
Descodificació de protocols	UART, CAN, SPI i I2C
Senyal mix en temps real	Sí
Màxima freqüència de mostreig lògica	40 MHz
Màxima freqüència de mostreig analògica	20 MHz
Màxim nombre de mostres	12000
Tipus de *trigger*	Comparador analògic i combinador lògic
Mode de *trigger*	Pujada, baixada i estat
Font d'alimentació	Connexió USB de 5 V
Rang de temperatures de funcionament	0 - 50 ºC
Resistent a l'aigua	Sí
Dimensions	20 x 110 x 8 mm
Pes	14 g
Software disponible	BitScope DSO i BitScope Logic

Taula 4.1 Especificacions tècniques del BitScope Micro (BS05U). Font: [2].

Al mateix temps, tal i com ja s'ha comentat, es disposa d'una biblioteca pel BitScope, anomenada BitLib, que és el que s'ha usat per realitzar aquest projecte. Les funcions que incorpora permeten seleccionar algunes de les especificacions anteriors. Per obtenir les dades a través d'aquestes funcions cal seguir el procés que ve descrit a continuació: primer

cal inicialitzar la biblioteca i el dispositiu, seguidament cal configurar-lo per poder usar-lo i capturar una trama, i així finalment obtenir les dades i tancar la biblioteca i el dispositiu.

5. Problemàtica a resoldre

Tal i com s'ha dit, l'objectiu d'aquest projecte és realitzar un analitzador de trames de CAN FD. Així doncs, per començar, cal adquirir els valors del senyal digital que surt d'un controlador o transceptor de CAN FD, cosa que es fa amb el ja esmentat BitScope Micro, de tal forma que cal configurar-lo i programar-lo adequadament amb llenguatge Python.

Un cop obtingudes les dades, cal processar-les i realitzar un algorisme per tal de fer les comprovacions adients per assegurar-se que són coherents, i poder identificar els bits corresponents.

Finalment, i a partir de les dades definitives, cal realitzar una interfície gràfica, per tal que l'usuari pugui introduir-hi uns valors d'entrada i obtingui un gràfic del senyal i informació del mateix. Això es realitza mitjançant Tkinter, que consisteix en una adaptació a llenguatge Python d'una biblioteca gràfica programada en un llenguatge diferent de Python [5]. S'escull aquesta biblioteca, ja que es poden obtenir resultats molt bons sense excessiva complicació, i s'adapta perfectament a les necessitats d'aquest projecte. Cal dir però, que si es volgués fer una interfície més potent segurament fóra més convenient utilitzar una altra biblioteca.

6. Solució implementada

Primer de tot, s'ha instal·lat la biblioteca BitLib subministrada pel fabricant del Bitscope, tal i com s'ha mencionat anteriorment. Tot seguit, s'ha hagut d'estudiar el funcionament del dispositiu mitjançant la biblioteca descrita i les seves característiques, i així, poder configurar-lo de la manera requerida per tal d'obtenir les dades corresponents. Així doncs, ha calgut triar el dispositiu, el canal en el qual s'ha connectat el senyal a estudiar, el mode lògic i la quantitat de mostres que es prendran, que seran el màxim possible. També cal triar la freqüència de mostreig (*Sample Rate*), però això es deixa a elecció de l'usuari. Aquest també ha d'introduir les velocitats de transmissió de les fases Arbitration Phase (*Arbitration-Phase Bit Rate*) i Data Phase (*Data-Phase Bit Rate*), ja siguin iguals o diferents. Cal indicar que tot això s'ha fet a partir del programa base del fabricant del dispositiu.

A continuació, per mitjà del BitScope Micro, s'adquireix un conjunt de mostres digitals del senyal a estudiar, de tal forma que es té més d'una mostra per cada bit. Per tal de que el dispositiu comenci a adquirir les dades quan es desitja, cal introduir-hi una condició, anomenada *trigger*, a part de que s'estableixi que el dispositiu rastregi el senyal d'entrada repetidament i de forma síncrona fins que no es compleixi aquesta condició. En aquest cas, com que el senyal és estable a 3,3 Volts, és a dir, quan es troba en el valor lògic alt, s'ha posat la condició d'inici d'adquisició de dades quan detecti que el senyal baixa i passa per 1,5 Volts. Hom es pot donar compte que mitjançant aquest mètode es pot induir a error, ja que pot ser que a l'hora d'executar el programa el dispositiu detecti un impuls que no sigui el primer. Això no serà així si s'executa primer el programa i després s'envia el senyal a estudiar. Per tal de detectar aquest error, tan sols cal observar si la trama és completada correctament.

Així doncs, s'obté un conjunt de mostres que es guarden en un fitxer assignant-los un valor en el temps segons la freqüència entrada per l'usuari, és a dir, el temps entre mostres és l'invers de la freqüència, i es comença a partir del temps zero. Aquest temps s'expressa en nanosegons per treballar amb més comoditat. Cal indicar que la funció d'adquisició de la biblioteca, per trames lògiques, retorna els valors zero i cinc. Aquest segon valor, a l'hora de guardar les mostres al fitxer, s'ha convertit en 1.

Tot seguit, es procedeix a l'obtenció dels bits corresponents a les dades adquirides, per la qual cosa cal conèixer les velocitats de transmissió de bits de les dues fases que seran els valors que introduirà l'usuari, com ja s'ha dit. Amb aquests valors, cal tenir en compte uns marges en ambdues fases degut als petits errors que es produeixen amb el dispositiu, que fa que les durades dels bits no siguin sempre exactes. Per tal d'identificar els bits de la Arbitration Phase el que es fa és mesurar la diferència de temps entre la primera mostra que té un valor i la primera que n'és diferent. Aquesta diferència es divideix per la durada del bit en aquesta fase, que és l'invers de la velocitat entrada per l'usuari, per així obtenir el nombre de bits

corresponents a aquesta diferència i al valor determinat. Aquest procés és iteratiu, es guarden els bits en una variable i es compten, fins que s'arriba al número de la llargada de l'Arbitration Phase. Ara bé, cal tenir en compte els possibles Stuff Bits, pel que es fa una funció auxiliar que, mentre encara no s'hagi assolit la llargada de la fase, compta cada cop el número que n'hi ha i si n'apareixen s'obtindran en total tants bits com la llargada de la fase més el nombre de Stuff Bits. A partir d'aquest punt es reprodueix el mateix mètode per a l'obtenció dels bits, calculant les diferències però ara amb l'altra velocitat de transmissió, la de la Data Phase. Cal indicar que amb aquest mètode no s'identifiquen bits de valor recessiu, que és l'estable, al final de la trama, pel que se li afegeixen cinc per si es dóna el cas.

Un cop obtinguda la trama de valors lògics, es procedeix a realitzar les comprovacions del protocol i la identificació dels bits. Primerament, s'eliminen els Stuff Bits que hi hagi fins a identificar els bits del DLC, amb els quals es calcula la longitud del Data Field i del CRC. Seguidament, s'eliminen tots els Stuff Bits fins a obtenir el Data Field i es guarda en una variable la quantitat trobada, per poder realitzar la comprovació del Stuff Bit Counter. Així doncs, en aquest punt ja es coneix la longitud de la trama sencera, i es poden identificar cadascun dels bits segons la seva posició explicada anteriorment. Aquí també es guarden les variables que es volen mostrar finalment a l'usuari i es converteixen de llenguatge binari a hexadecimal. Al mateix temps, es comproven que els Fixed Stuff Bits (FSBs) siguin de valor oposat al seu bit anterior, es comprova el bit de paritat i el CRC. Això últim es fa per mitjà d'unes funcions en Python, en què cal entrar-hi el polinomi generador, el valor inicial, la longitud del CRC, la part de la trama que s'usa per fer la divisió i la llargada d'aquesta.

A continuació, en la Figura 6.1, es pot observar, de forma general, el procés seguit en la solució implementada, sense entrar molt en detall degut a la complicació de l'algorisme elaborat. Si és d'interès conèixer-lo en profunditat, es recomana endinsar-se en el codi en Python de l'Annex A.

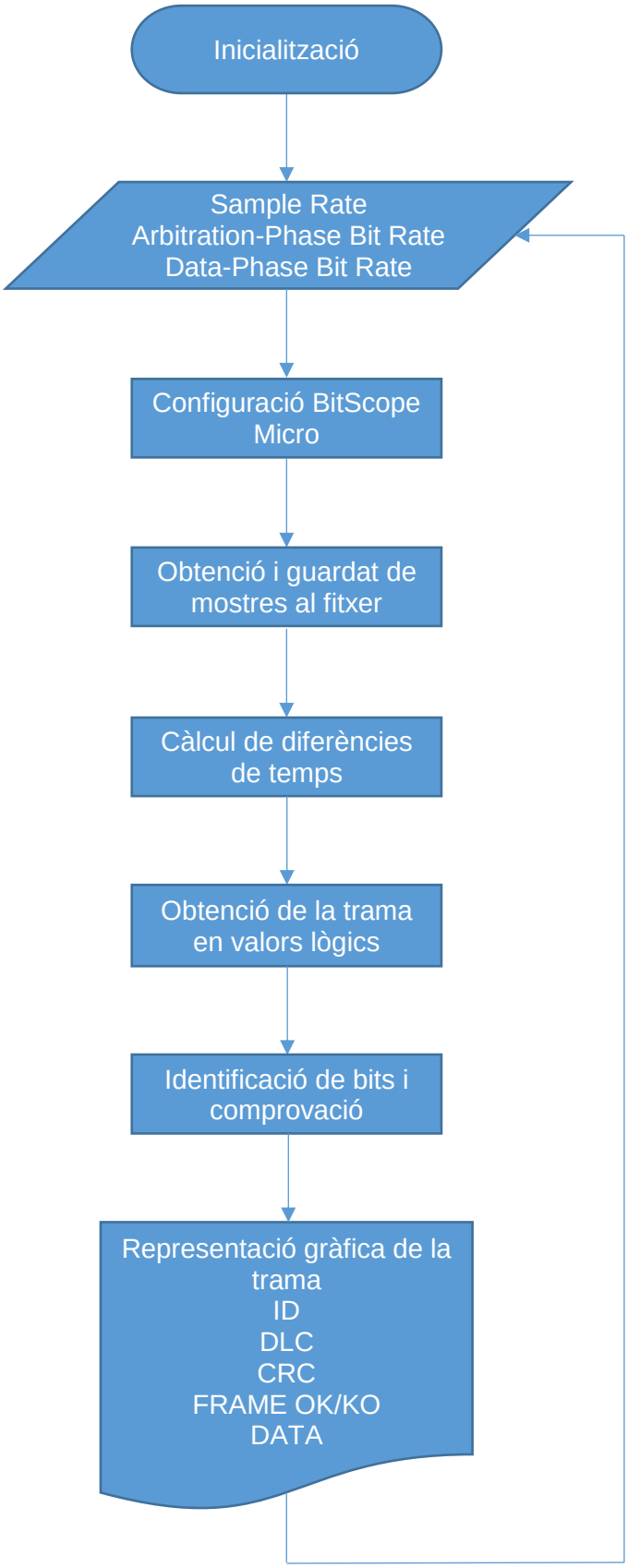

Figura 6.1 Diagrama de blocs general de la solució implementada. Font: pròpia.

Finalment, per mostrar els resultats, tal i com ja s'ha dit, es crea una interfície gràfica mitjançant el Tkinter, on l'usuari introdueix les dades d'entrada descrites (*Inputs*) i prement un botó (*Capture*) es retorna la informació de la trama captada (*Information*). Això està fet de tal forma que quan es prem el botó, es fa la configuració del BitScope Micro, l'adquisició de dades i la identificació dels paràmetres de la trama. Com es pot veure en la Figura 6.2, a l'usuari es mostra, a part de la representació gràfica de la trama realitzada amb la biblioteca Matplotlib [6], l'identificador de la trama, el DLC, el CRC, la Data i si la trama és correcte (*OK*) o no (*KO*), això és, si totes les comprovacions són correctes. A més a més, a la part inferior també s'incorpora un conjunt d'eines preestablertes d'aquesta interfície.

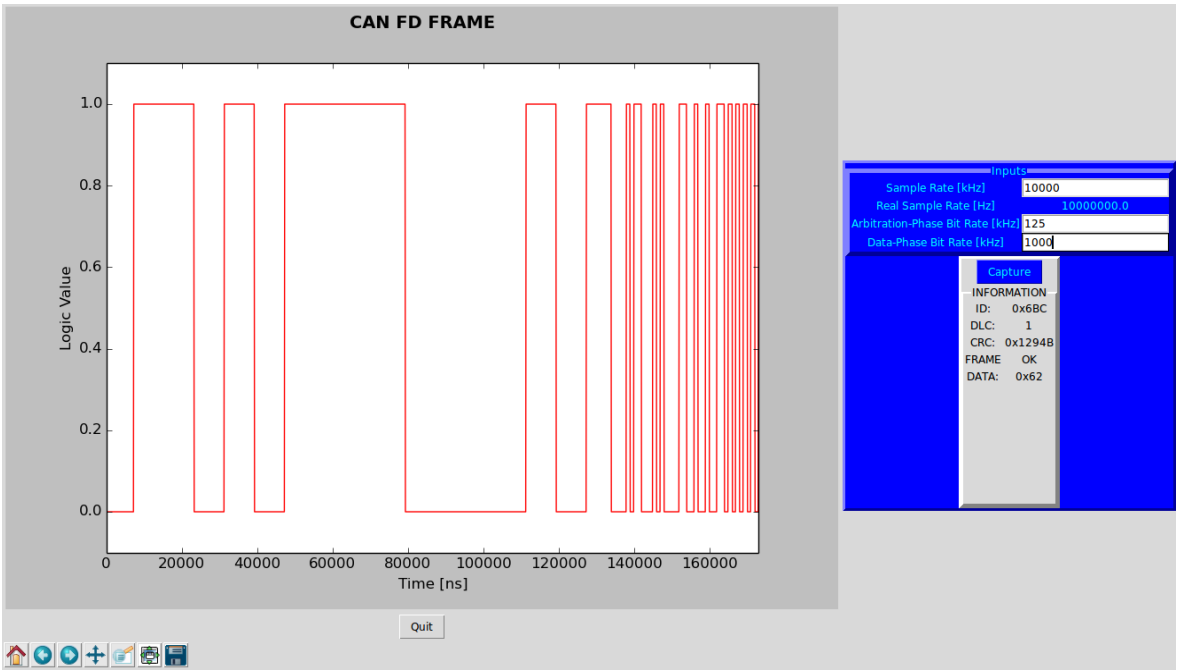

Figura 6.2 Interfície gràfica mostrant una trama. Font: pròpia.

7. Problemes trobats

El primer problema trobat va ser la escassa documentació de les funcions de la biblioteca BitLib, ja que per algunes no estava especificat com utilitzar-les ni quins valors podien prendre, per la qual cosa l'única manera de conèixer-les era fent proves. Per altres, mirant l'arxiu on es troba la biblioteca es van poder entendre millor. Un cas concret és el de la funció per seleccionar el canal on està connectat el senyal, ja que només s'indica que si es vol triar analògic és del 0 al 3 i si es vol lògic del 4 al 11, però no s'indica la correlació d'aquests valors amb el lloc físic d'entrada. Provant es va concloure que per a la connexió L0 cal seleccionar el canal 4. Un altre cas és el de la funció *trigger*, ja que en la documentació no s'indica les diferents possibilitats de condicions que es poden posar. Per conèixer-ho, va ser necessari consultar l'arxiu font de la biblioteca.

Un segon problema que ja s'ha esmentat és que la durada dels bits no és exacte, cosa que inicialment feia que no s'agafés la trama correctament, pel que va ser necessari establir-hi uns marges d'error. D'aquesta forma, en la major part dels casos l'algoritme funciona com és degut, però no sempre, i, segons sembla després d'un conjunt d'experiments que es poden veure a continuació, per velocitats de transmissió elevades, és quan es produeix més error i la trama no es capta correctament.

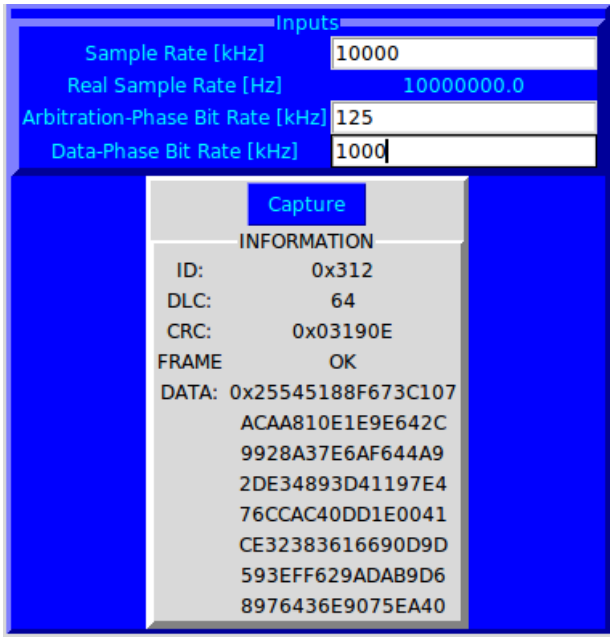

Figura 7.1 Experiment a 125 i 1000 kbit/s. Font: pròpia.

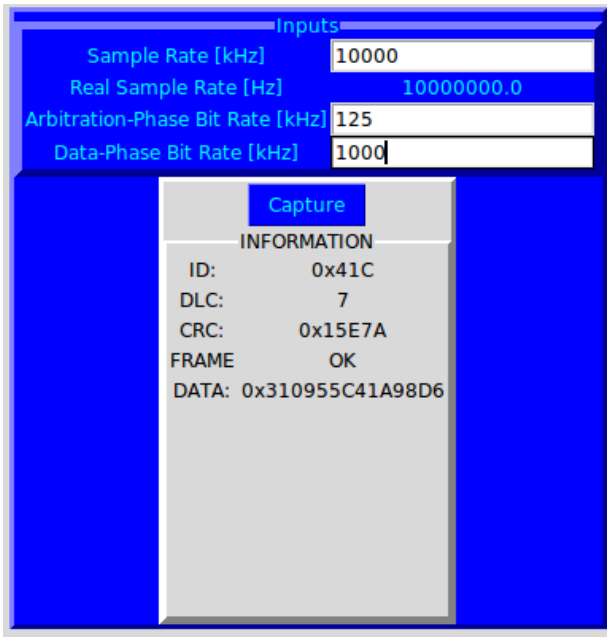

Figura 7.2 Experiment a 125 i 1000 kbit/s. Font: pròpia.

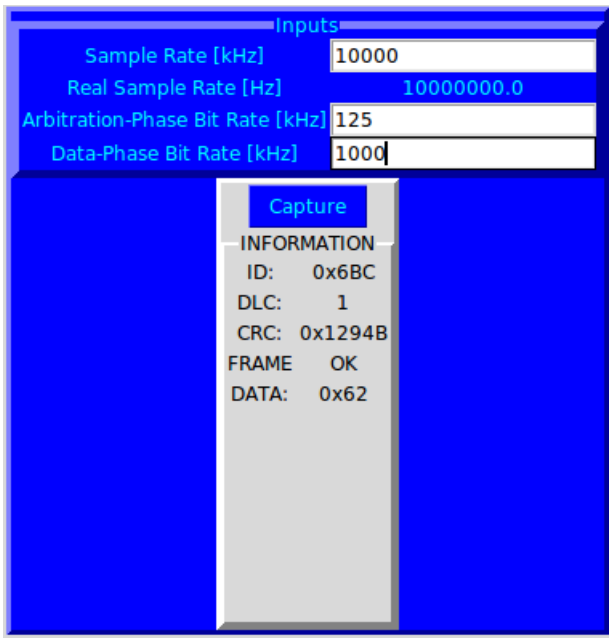

Figura 7.3 Experiment a 125 i 1000 kbit/s. Font: pròpia.

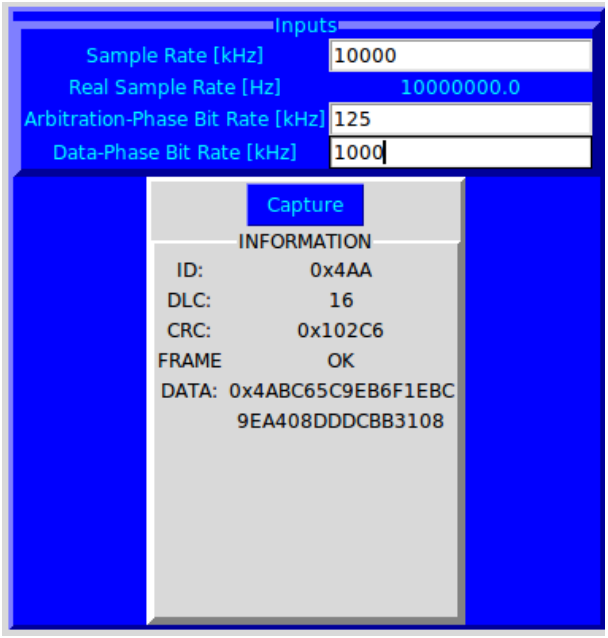

Figura 7.4 Experiment a 125 i 1000 kbit/s. Font: pròpia.

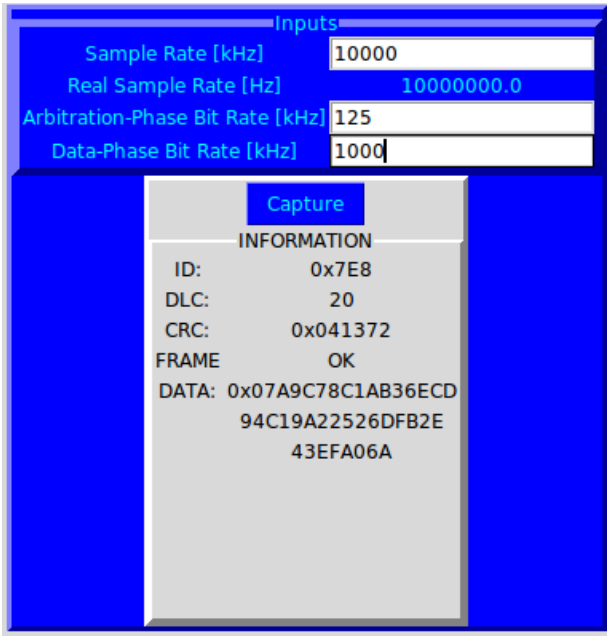

Figura 7.5 Experiment a 125 i 1000 kbit/s. Font: pròpia.

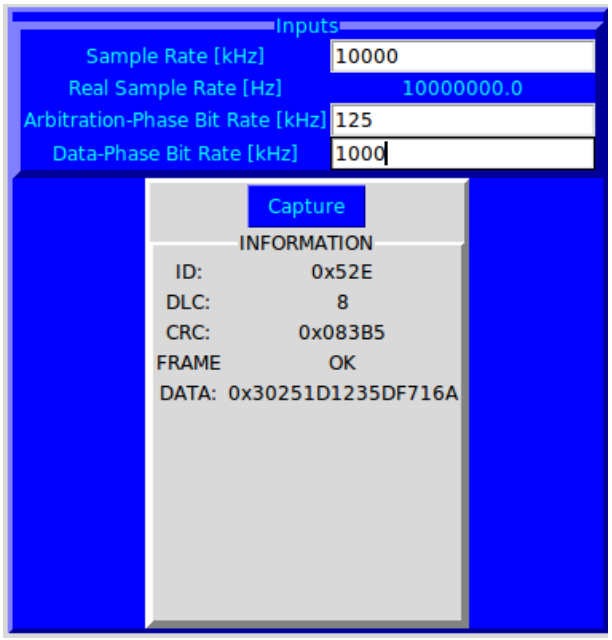

Figura 7.6 Experiment a 125 i 1000 kbit/s. Font: pròpia.

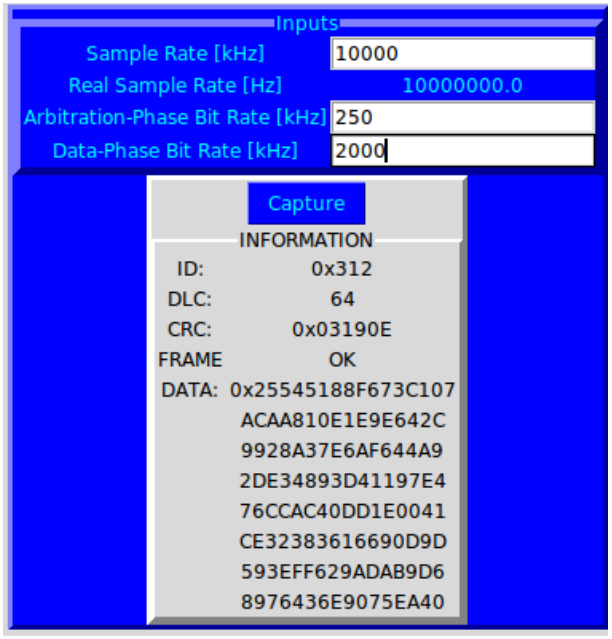

Figura 7.7 Experiment a 250 i 2000 kbit/s. Font: pròpia.

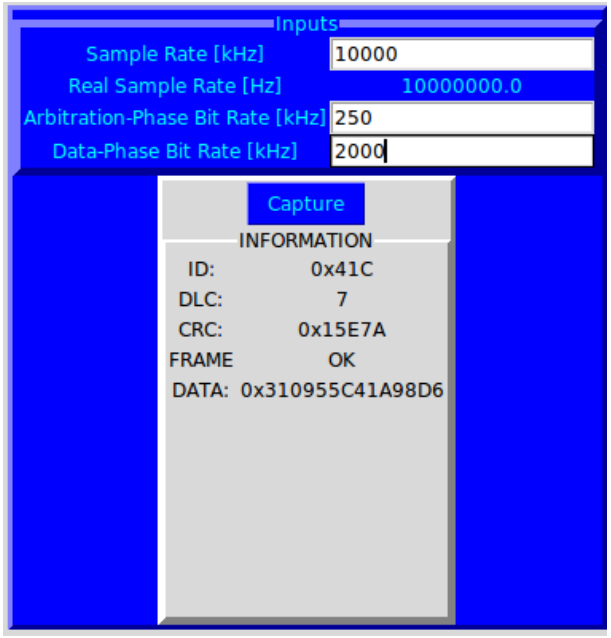

Figura 7.8 Experiment a 250 i 2000 kbit/s. Font: pròpia.

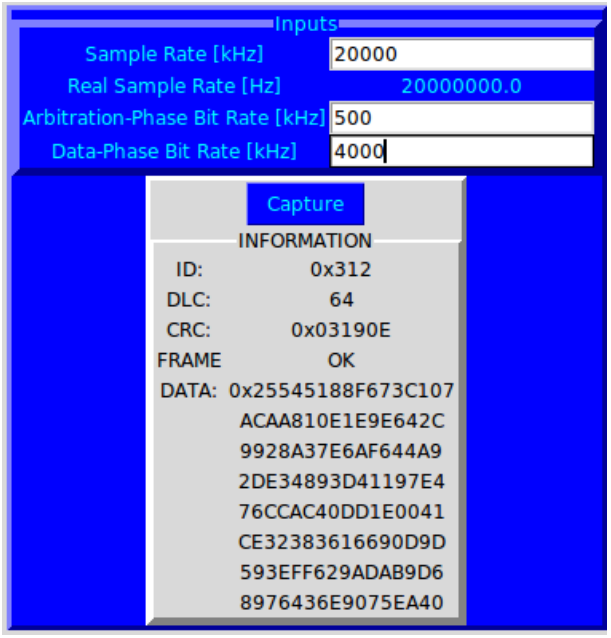

Figura 7.9 Experiment a 500 i 4000 kbit/s. Font: pròpia.

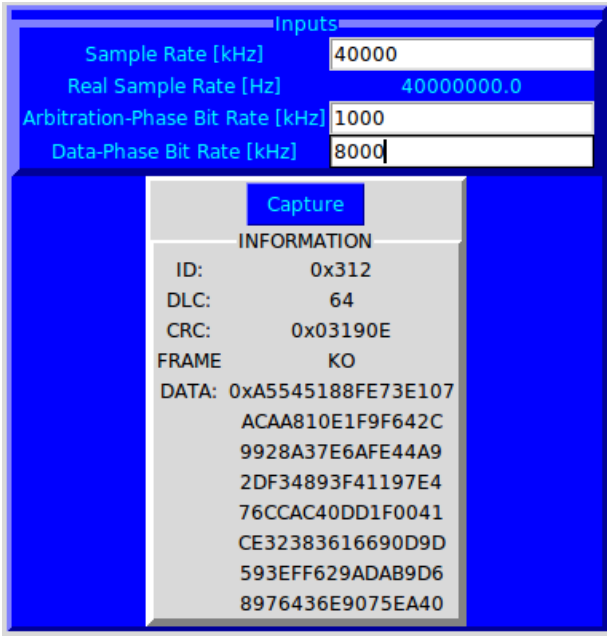

Figura 7.10 Experiment a 1000 i 8000 kbit/s. Font: pròpia.

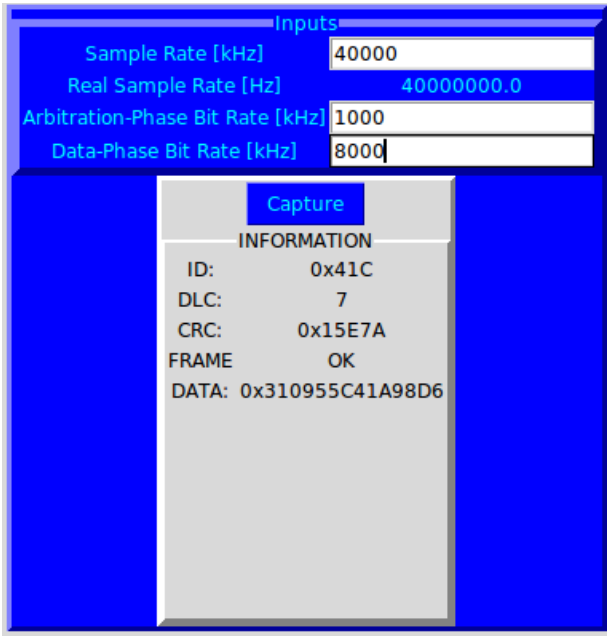

Figura 7.11 Experiment a 1000 i 8000 kbit/s. Font: pròpia.

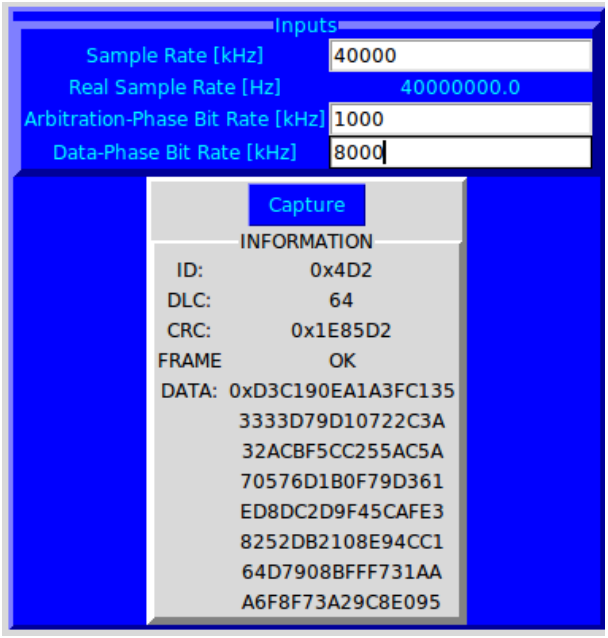

Figura 7.12 Experiment a 1000 i 8000 kbit/s. Font: pròpia.

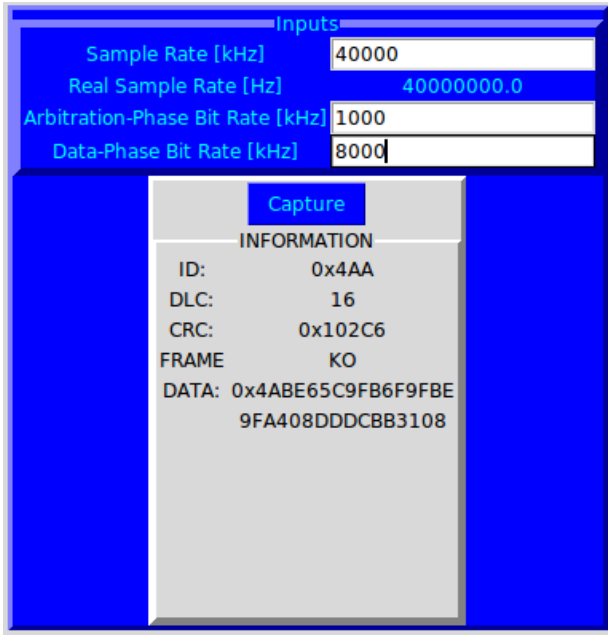

Figura 7.13 Experiment a 1000 i 8000 kbit/s. Font: pròpia.

Tal i com es pot veure en les Figures 7.1, 7.2, 7.3, 7.4, 7.5 i 7.6, totes elles amb els mateixos valors entrats, l'algoritme funciona perfectament per velocitats de transmissió baixes. Si s'augmenten una mica aquestes velocitats, com en les Figures 7.7 i 7.8, segueix funcionant correctament, i inclús si s'augmenten encara més, com en la Figura 7.9, també és així. Pels valors entrats en aquest darrer experiment, degut a que només s'ha fet un cop, podria ser que en altres casos ja comencés a fallar, ja que els següents experiments amb els valors

augmentats ja s'observen alguns errors. Això es veu en les Figures 7.10, 7.11, 7.12 i 7.13, en què es tracten trames amb les màximes velocitats, i dos casos s'han detectat adientment però els altres dos no. Cal indicar que a mesura que s'han anat augmentant les velocitats també s'ha augmentat la freqüència de mostreig, ja que sinó no és possible captar les trames correctament.

Si s'observa amb atenció la Figura 7.10 en què ha resultat *KO*, es veu que és la mateixa trama captada a velocitats inferiors en les Figures 7.1, 7.7 i 7.9, i el que succeeix és que la *DATA* és una mica diferent, i això és culpa de les durades incorrectes dels bits. El mateix passa amb la trama de la Figura 7.13 i la seva corresponent captada a velocitats inferiors que és la Figura 7.4. Per comprovar aquest problema es va fer servir l'analitzador de trames BitScope Logic introduït anteriorment, on es veia que alguns bits duraven a vegades més i a vegades menys de l'esperat.

Un tercer problema és que, usant la funció *trigger*, la durada del primer bit es redueix al voltant d'uns sis-cents nanosegons, i, per tal d'aconseguir la trama correcta, l'única manera trobada de resoldre-ho va ser sumant-li aquest temps en el primer càlcul. Per deduir que el problema era aquesta funció, es va fer una prova de programar el dispositiu en llenguatge C, que es pot veure en l'Annex B, i succeïa el mateix. També es va fer servir el BitScope Logic, que també permet introduir el *trigger* i on es pot veure unes mostres abans d'aquest, i s'observava que el primer bit tenia la durada esperada però el *trigger* funcionava amb un petit retard. Aquest problema es va provar d'arreglar usant una altra funció de la biblioteca, la qual permet agafar dades prèvies a la condició establerta pel *trigger* com en el BitScope Logic, però es va descobrir que utilitzant aquesta funció la freqüència de mostreig es modificava, cosa que no interessava i per això es va descartar.

Per posar un exemple d'aquest problema, a continuació es poden veure les primeres mostres de l'arxiu d'una trama de velocitats de 1 Mbit/s i 8 Mbit/s, amb freqüència de mostreig de 40 Mbit/s. S'observa que el primer bit, amb valor 0, té una durada de 375 ns, quan hauria de ser 1000, i en canvi, el següent bit sí que té aquesta durada, i el seus successius també.

0.0,0
25.0,0
50.0,0
75.0,0
100.0,0
125.0,0
150.0,0
175.0,0
200.0,0
225.0,0
250.0,0
275.0,0
300.0,0

325.0,0
350.0,0
375.0,1
400.0,1
425.0,1
450.0,1
475.0,1
500.0,1
525.0,1
550.0,1
575.0,1
600.0,1
625.0,1
650.0,1
675.0,1
700.0,1
725.0,1
750.0,1
775.0,1
800.0,1
825.0,1
850.0,1
875.0,1
900.0,1
925.0,1
950.0,1
975.0,1
1000.0,1
1025.0,1
1050.0,1
1075.0,1
1100.0,1
1125.0,1
1150.0,1
1175.0,1
1200.0,1
1225.0,1
1250.0,1
1275.0,1
1300.0,1
1325.0,1
1350.0,1
1375.0,0
1400.0,0
1425.0,0
1450.0,0
1475.0,0
1500.0,0
1525.0,0
1550.0,0
1575.0,0

1600.0,0
1625.0,0
1650.0,0
1675.0,0
1700.0,0
1725.0,0
1750.0,0
1775.0,0
1800.0,0
1825.0,0
1850.0,0
1875.0,0
1900.0,0
1925.0,0
1950.0,0
1975.0,0
2000.0,0
2025.0,0
2050.0,0
2075.0,0
2100.0,0
2125.0,0
2150.0,0
2175.0,0
2200.0,0
2225.0,0
2250.0,0
2275.0,0
2300.0,0
2325.0,0
2350.0,0
2375.0,0
2400.0,0
2425.0,0
...
...
(mostres amb bit de valor 0)
...
...
6100.0,0
6125.0,0
6150.0,0
6175.0,0
6200.0,0
6225.0,0
6250.0,0
6275.0,0
6300.0,0
6325.0,0
6350.0,0
6375.0,1

6400.0,1
6425.0,1
6450.0,1
6475.0,1
6500.0,1
6525.0,1
6550.0,1
6575.0,1
6600.0,1
...

Així doncs, si calculem les diferències de temps quan canvia el valor del bit s'obtenen les durades següents:

- 375 ns a valor 0.
- 1000 ns a valor 1.
- 5000 ns a valor 0.
- Indeterminats ns a valor 1.

Amb questes diferències, seguint el mètode descrit, s'obtenen el conjunt de valors lògics 01000001. Aquests bits s'identifiquen com el dominant del SOF i amb un inici de Base Identifier d'un 4 i seguit d'un 0 o un 1 en funció de les pròximes mostres, ja que el darrer bit recessiu que s'observa del tros de la trama és un Stuff Bit. Això es correspon amb el resultat obtingut mitjançant l'algoritme i mostrat en la interfície visible en la Figura 7.14, on es veu que l'identificador (*ID*) té el valor 41C en hexadecimal. D'aquesta forma, es demostra que només el primer bit té una durada incorrecta, pel que sumar-li sis-cents nanosegons és una solució adequada, tenint sempre en compte el marge introduït.

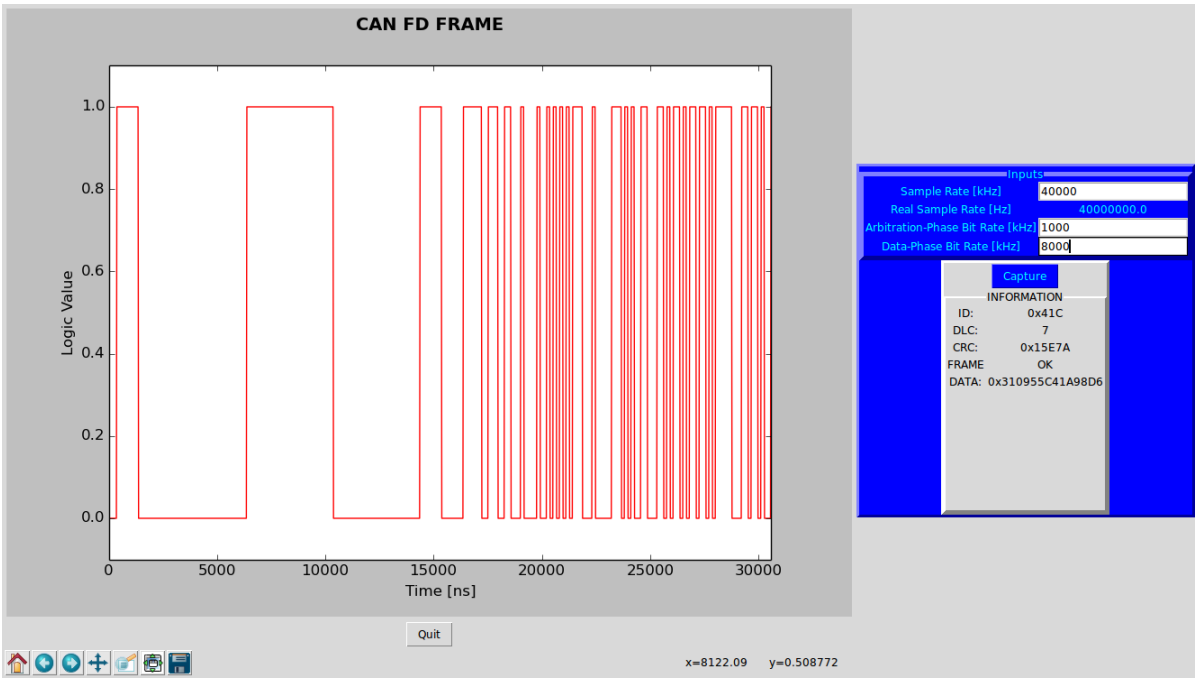

Figura 7.14 Interfície gràfica mostrant la trama de l'exemple. Font: pròpia.

Un altre problema, que de fet és una característica del BitScope Micro, és que no permet qualsevol freqüència de mostreig, i si s'introdueix una no disponible aquest selecciona la més propera permesa. És per això que a l'usuari se li mostra la freqüència real triada (*Real Sample Rate*), tal i com es pot veure en la Figura 7.14.

8. Futures millores

Una millora que es podria fer per aquest projecte és que servís també per senyals de CAN FD de format estès, o sigui els que tenen vint-i-nou bits d'identificador. Això és perquè de la manera en què està elaborada el codi només s'identifiquen correctament les trames de format base o estàndard. Per fer-ho, caldria tenir en compte el valor del bit IDE, i identificar la resta de bits a partir d'aquest, cosa que resultaria un mica laboriosa.

També es podria millorar la part de l'algoritme en què es detecta el bit BRS, que per definició té una durada diferent de tota la resta. En els casos tractats, aquest sempre té valor diferent del seu bit següent, el ESI, però si no fos així, no s'identificaria correctament.

Per últim, pel que fa a la interfície gràfica, també es podria millorar i fer més atractiva per a l'usuari, ja que tal i com està actualment té la característiques mínimes necessàries. A la vegada, també es podria optimitzar tot el codi.

9. Planificació i programació

Si bé és cert que pel tipus de projecte no es poden definir diferents etapes seguides amb detall, sí que se'n poden distingir tres parts. La primera, l'estudi del protocol de comunicació CAN FD. La segona, la comprensió del funcionament del BitScope Micro i la seva configuració. I la tercera i definitiva, la programació de l'algorisme i la presentació de les dades en la interfície gràfica.

10. Pressupost

Els costos més rellevants d'aquest projecte queden definits en la Taula 10.1.

	Quantitat	Preu/unitat	Total
BitScope Micro	1	145 $	125 €
Enginyer	300 h	40 €/h	12000 €
Amortització ordinador	1	60 €	60 €
			12185 €

Taula 10.1 Relació de costos del projecte (conversió de dòlars a euros a dia 14/06/2018).

Com es pot observar el cost total no és molt elevat, ja que el projecte s'ha centrat en la programació. Si s'implementés en una Raspberry Pi, s'hauria afegir-hi el seu cost, també reduït, d'uns 40 €.

Cal indicar que pel càlcul de l'amortització de l'ordinador s'ha tingut en compte que s'ha fet servir durant sis mesos, un ordinador del valor de sis-cents euros i amb una vida útil de cinc anys, el que resulta un cost de seixanta euros. També es fa notar que les plaques usades per reproduir els senyals de CAN FD són aprofitades d'altres projectes, pel que no han suposat un cost nou per aquest.

11. Impacte ambiental

De la mateixa manera que succeeix amb el pressupost, com que el projecte s'ha centrat en la programació, l'impacte ambiental és bastant reduït, tan sols cal tenir en compte el produït pel BitScope Micro. Durant la seva vida útil no produeix cap impacte ambiental, però sí quan acaba. Com que està considerat com equipament elèctric o electrònic (EEE), no es pot dipositar en un contenidor habitual de brossa, pel que s'ha de sotmetre a la recollida selectiva.

Ara bé, l'ús del CAN FD sí que tindrà en els propers anys un impacte positiu, ja que farà les comunicacions més ràpides i eficients, amb la conseqüent reducció de consum d'energia.

Conclusions

Com a conclusió es pot dir que s'ha aconseguit fer l'analitzador de trames CAN FD pretès, amb algunes millores pendents de realitzar per optimitzar el seu funcionament. També s'ha acomplert amb l'objectiu de comprendre el protocol de comunicació CAN FD. I pel que fa a la interfície gràfica, s'ha aconseguit configurar-la i ha resultat completament útil i suficient per la necessitat del projecte.

A la vegada, es pot concloure que, tot i les grans facilitats que proporciona el BitScope Micro, té inconvenients també, i alguns d'ells s'han aconseguit resoldre eficaçment, però d'altres de manera menys rigorosa. No es pot descartar que, per falta de documentació, no s'hagi configurat correctament el dispositiu i això provoqui els problemes descrits.

Finalment, es conclou que el projecte està llest per acoblar-se a una Raspberry Pi i que així esdevingui portàtil. Tan sols cal instal·lar-hi la biblioteca BitLib, introduir-hi el codi i realitzar les connexions adients.

Agraïments

Vull agrair el suport rebut per part del tutor Manuel Moreno Eguílaz, sempre disponible a assistir-me quan li ho he requerit i que m'ha ajudat a entendre el protocol de comunicació CAN FD i a resoldre els problemes trobats al llarg del projecte.

També vull donar les gràcies als professors del grau que m'han format, i en concret els dels departaments d'informàtica i electrònica, ja que els seus ensenyaments m'han servit especialment per completar aquest treball.

Finalment, vull expressar el meu enorme agraïment a la meva família i als meus amics, que sense el suport d'aquests, ja no només en aquest projecte sinó en els quatre anys del grau, tot plegat hagués esdevingut molt més feixuc.

Bibliografia

[1] ROBERT BOSCH GMBH. *CAN with Flexible Data-Rate,* Alemanya, 2012.

[2] BITSCOPE. *Mixed Signal Solutions.* [http://www.bitscope.com/, 16 de juny de 2018].

[3] CAN IN AUTOMATION. *CAN FD - The basic idea.* [https://www.can-cia.org/can-knowledge/can/can-fd/, 16 de juny de 2018].

[4] DR. MUTTER, A., ROBERT BOSCH GMBH. *CAN FD and the CRC issue.* Alemanya, 2015.

[5] PYTHON SOFTWARE FOUNDATION. *Graphical User Interfaces with Tk.* [https://docs.python.org/2/library/tk.html, 16 de juny de 2018].

[6] THE MATPLOTLIB DEVELOPMENT TEAM. *Embedding in Tk.* [https://matplotlib.org/gallery/user_interfaces/embedding_in_tk_sgskip.html#sphx-glr-gallery-user-interfaces-embedding-in-tk-sgskip-py, 16 de juny de 2018].